STORM

流式数据处理

by xiaofei

概念

Storm is a free and open source distributed realtime computation system.

描述

Storm makes it easy to reliably process unbounded streams of data, doing for **realtime processing** what Hadoop did for **batch processing**. Storm is simple, can be used with any programming language, and is a lot of fun to use!

功能特点

便于集成

Spout已实现与Kestrel、RabbitMQ/AMQP、Kafka、JMS等的集成插件

开发简单易于使用

只需要实现Spout、Bolt和topology,将topology提交到集群即可

水平扩展

可通过增加机器扩展Spout和Bolt的处理能力

保障数据被处理

- 记录级容错
- Transction

容错

- **worker死掉了:**Supervisor将重启它，持续fail且无心跳到Nimbus，Nimbus将重新分配它到其它机器。
- **node死掉了:**task分配到的机器无响应或time out，Nimbus将分配到其它机器
- **Nimbus或Supervisor服务死掉了:**Nimbus和Supervisor被设计为了无状态的，所有的状态都位于Zookeeper和本地磁盘。只要用daemontools或monit做好进程的监控，失败会就会自动重启。
- **Nimbus是否单点(SPOF):**Nimbus失效后不会影响Supervisor监管Worker，但会影响Worker的重新分配。所以这里需要着重增加监管。

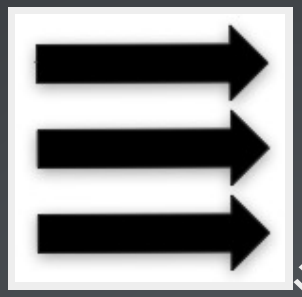
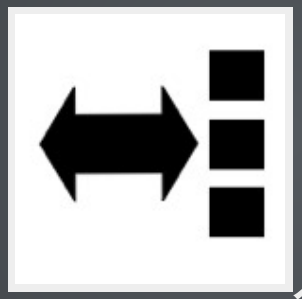
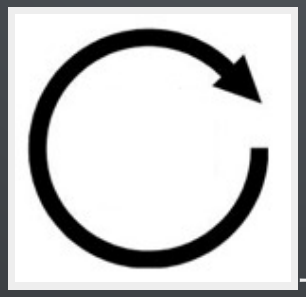
易于部署和运维

- 架构类似于MapReduce，便于运维人员维护

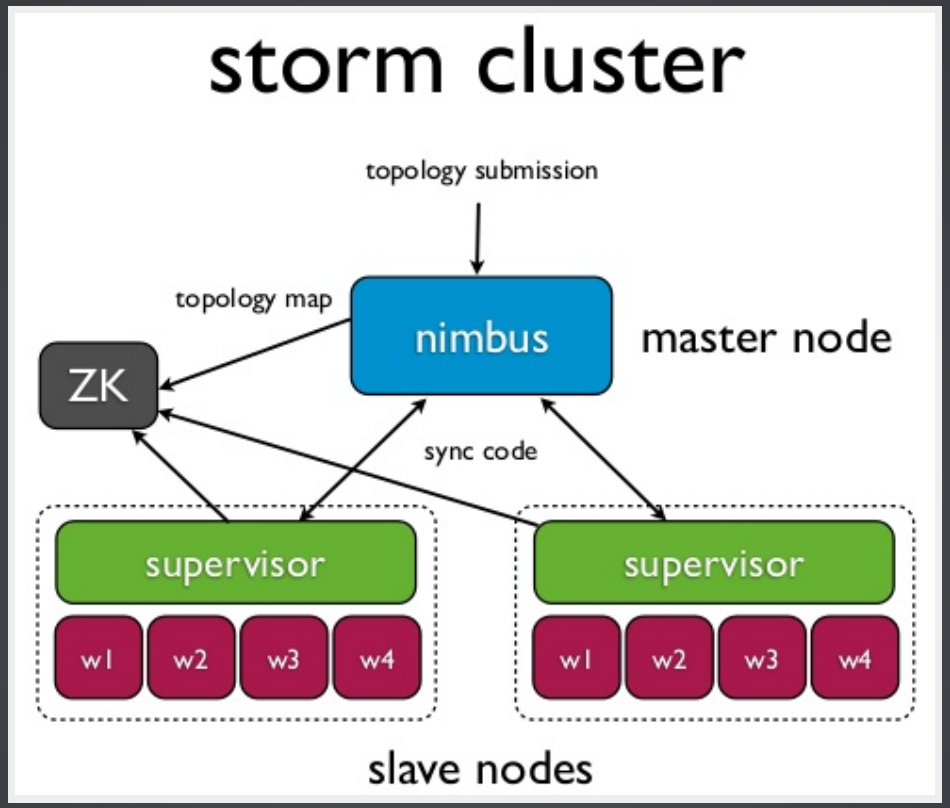
兼容多语言

Java、Scala、JRuby、Jython、Clojure..

使用场景

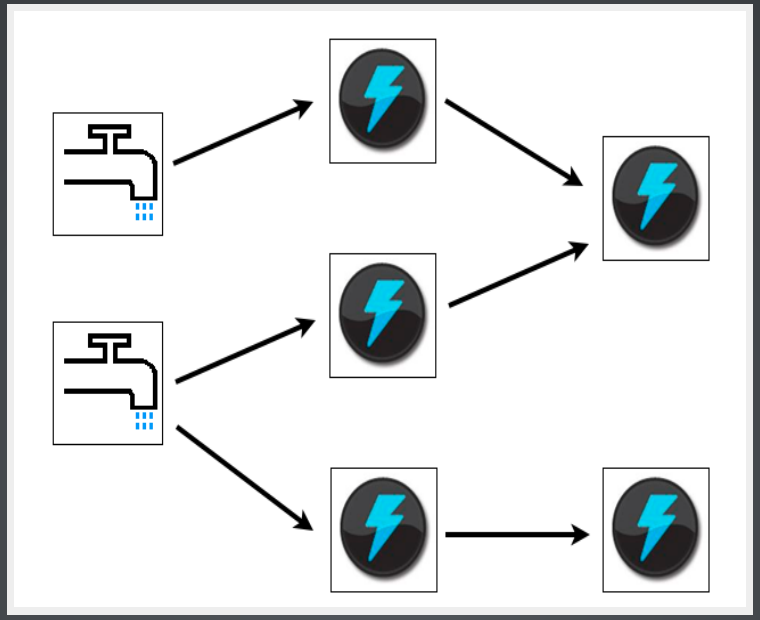
-  流式处理
-  分布式RPC
-  持续计算

系统构架


storm cluster
topology submission
topology map
nimbus
master node
ZK
sync code
supervisor
supervisor
w1
w2
w3
w4
w1
w2
w3
w4
slave nodes

Hadoop	Storm
JobTracker	Nimbus
	Zookeeper
TaskTracker	Supervisor
Task	Worker

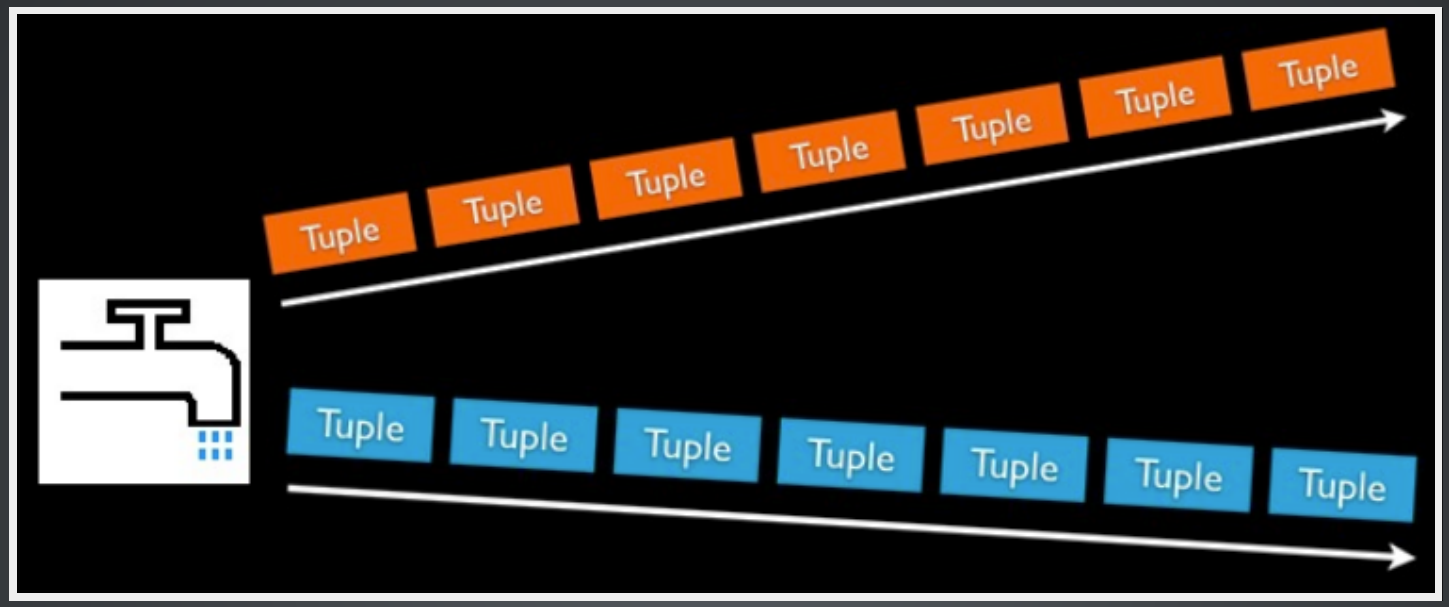
编程模型



Hadoop	**Storm**
Job	Topology
Map	Spout
Reduce	Bolt
Writable	Tuple
Task(Map/Reduce)	Task(Spout/Bolt)

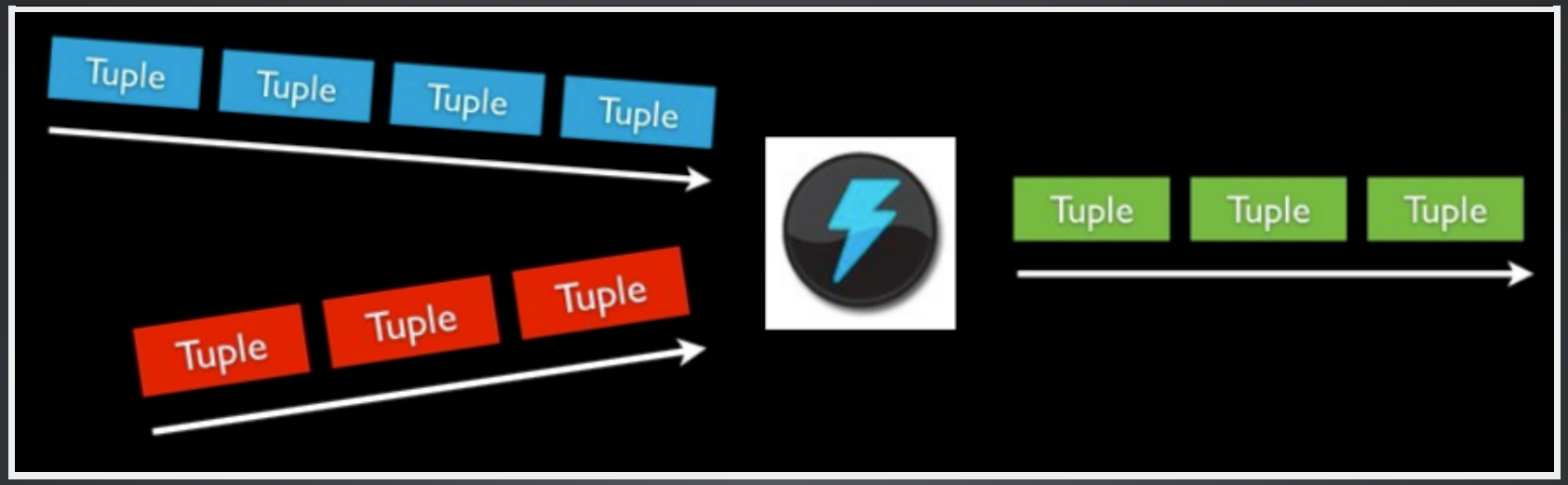
STREAMS

SPOUTS




```
public interface ISpout extends Serializable {
    void open(Map conf, TopologyContext context,
       SpoutOutputCollector collector);
    void close();
    void nextTuple();
    void ack(Object msgId);
    void fail(Object msgId);
}
```

BOLTS

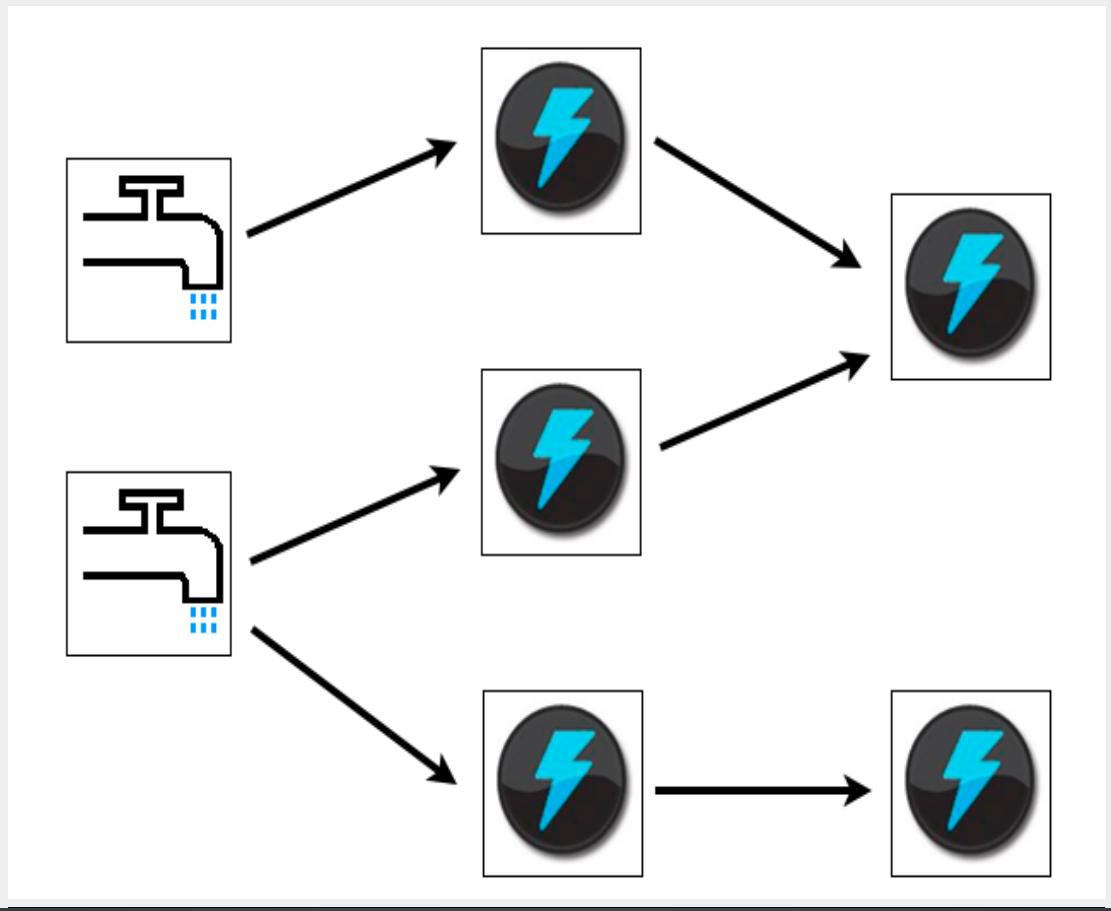


- 功能性操作
- 过滤操作
- 聚集操作
- Join操作
- 持久化操作(DB/Cache/HBase/Cassandra)

BOLTS

```
public interface IBolt extends Serializable {
    void prepare(Map stormConf, TopologyContext context, OutputCollector
    void execute(Tuple input);
    void cleanup();
}
```

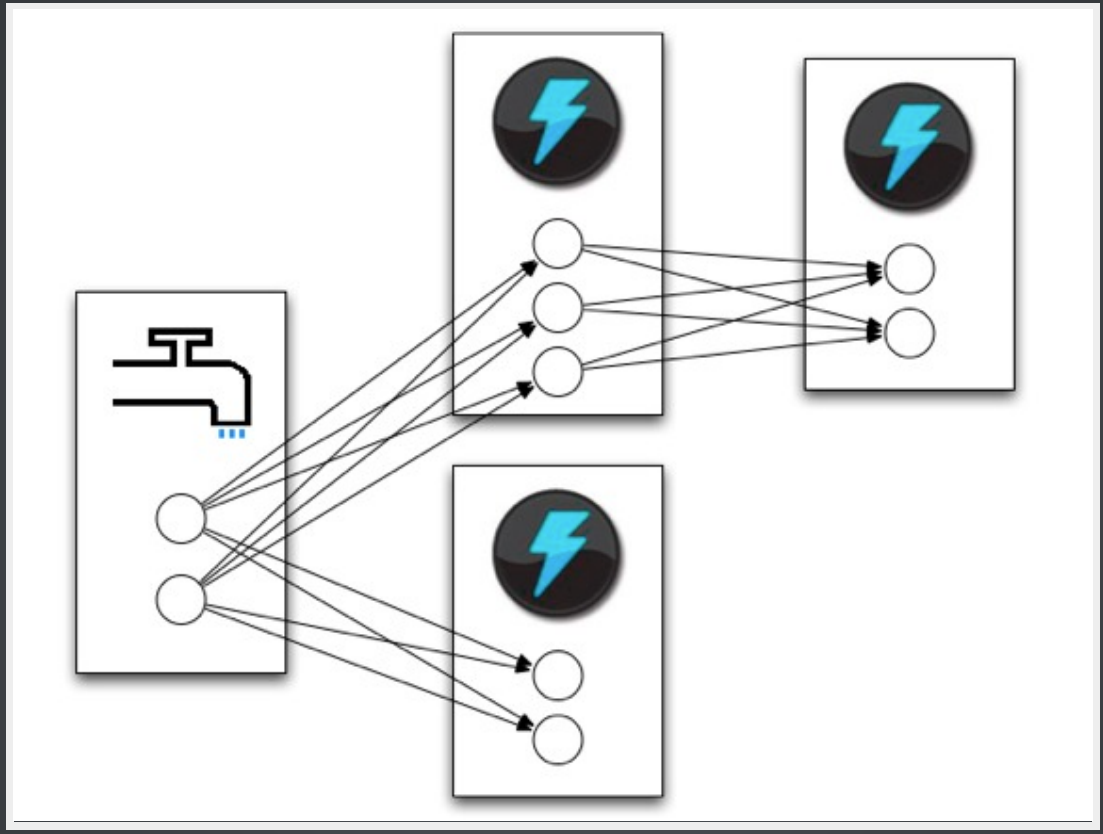
TOPOLOGIES



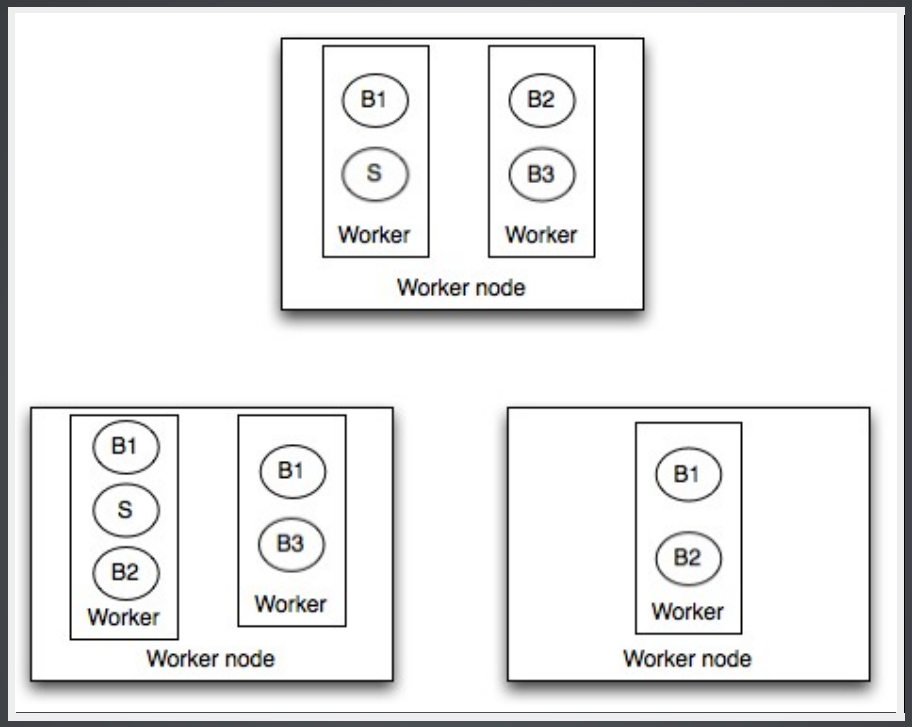
TOPOLOGIES

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("sentences", new KestrelSpout("kestrel.backtype.com",
                                               22133,
                                               "sentence_queue",
                                               new StringScheme()));
builder.setBolt("split", new SplitSentence(), 10)
        .shuffleGrouping("sentences");
builder.setBolt("count", new WordCount(), 20)
        .fieldsGrouping("split", new Fields("word"));
```

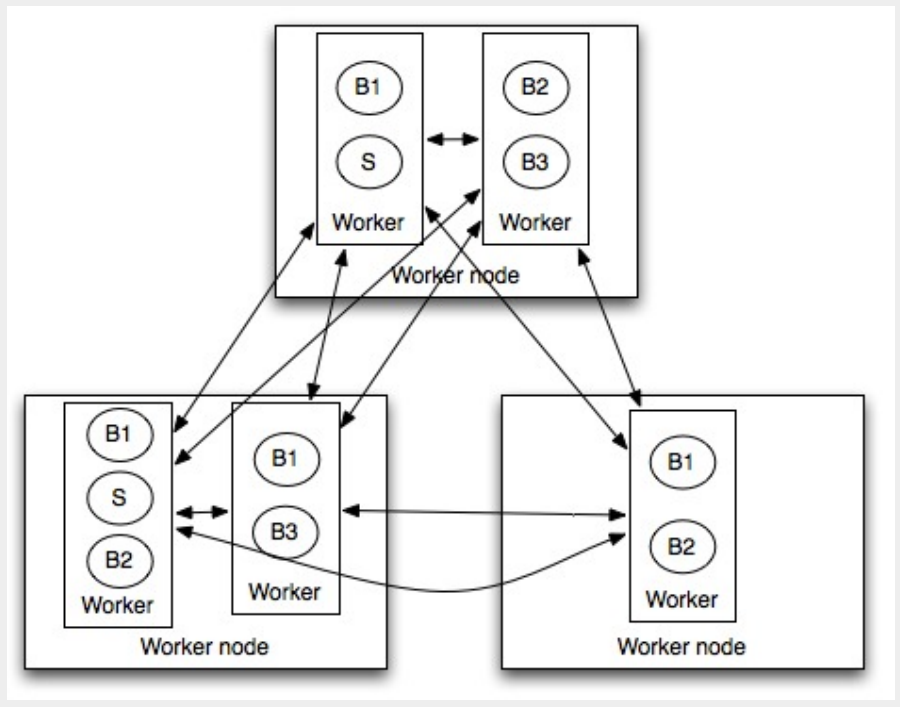
TASKS



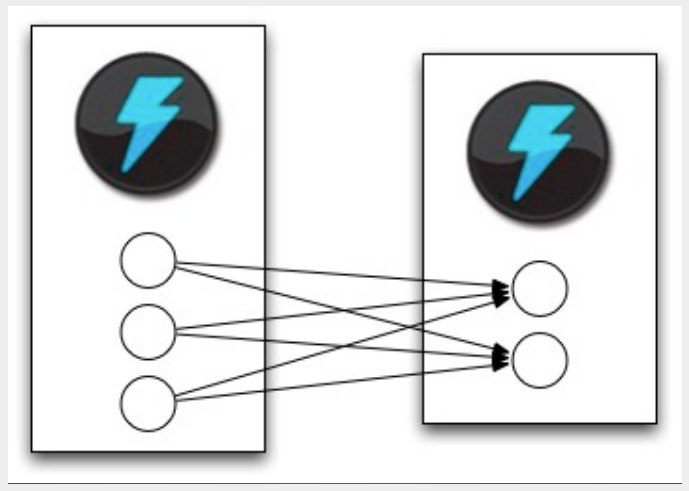
任务执行


B1
S
Worker
B2
B3
Worker
Worker node
B1
S
B2
Worker
B1
B3
Worker
Worker node
B1
B2
Worker
Worker node

任务执行

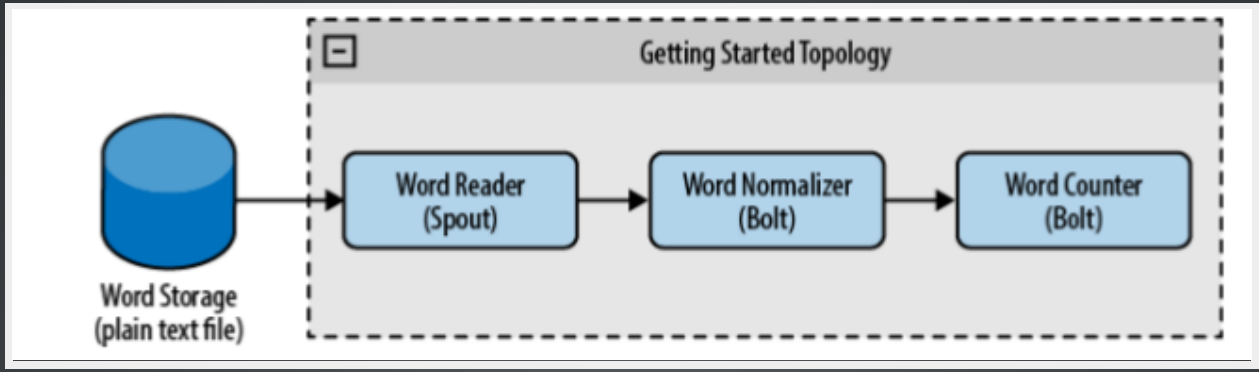

B1
S
Worker
B2
B3
Worker
Worker node
B1
S
B2
Worker
B1
B3
Worker
Worker node
B1
B2
Worker
Worker node

数据流分组



- Shuffle grouping:选择随机任务
- Fields Grouping:在tuple某字段上hash后求模
- All grouping:发送到所有的任务上
- Global grouping:取最小id的任务
- Direct:发送消息时指定任务
- Custorm:用户自定义,实现CustormStreamGrouping

WORD COUNT


Getting Started Topology
Word Reader
(Spout)
Word Normalizer
(Bolt)
Word Counter
(Bolt)
Word Storage
(plain text file)

WORDREADER

```
public class WordReader implements IRichSpout {
  private SpoutOutputCollector collector;
  private FileReader fileReader;
  private boolean completed = false;
  private TopologyContext context;
  public boolean isDistributed() {return false;}
  public void ack(Object msgId) {
    System.out.println("OK:"+msgId);
  }
  public void close() {}
  public void fail(Object msgId) {
    System.out.println("FAIL:"+msgId);
  }
  /**
  * The only thing that the methods will do It is emit each
  * file line
  */
```

WORDNORMALIZER

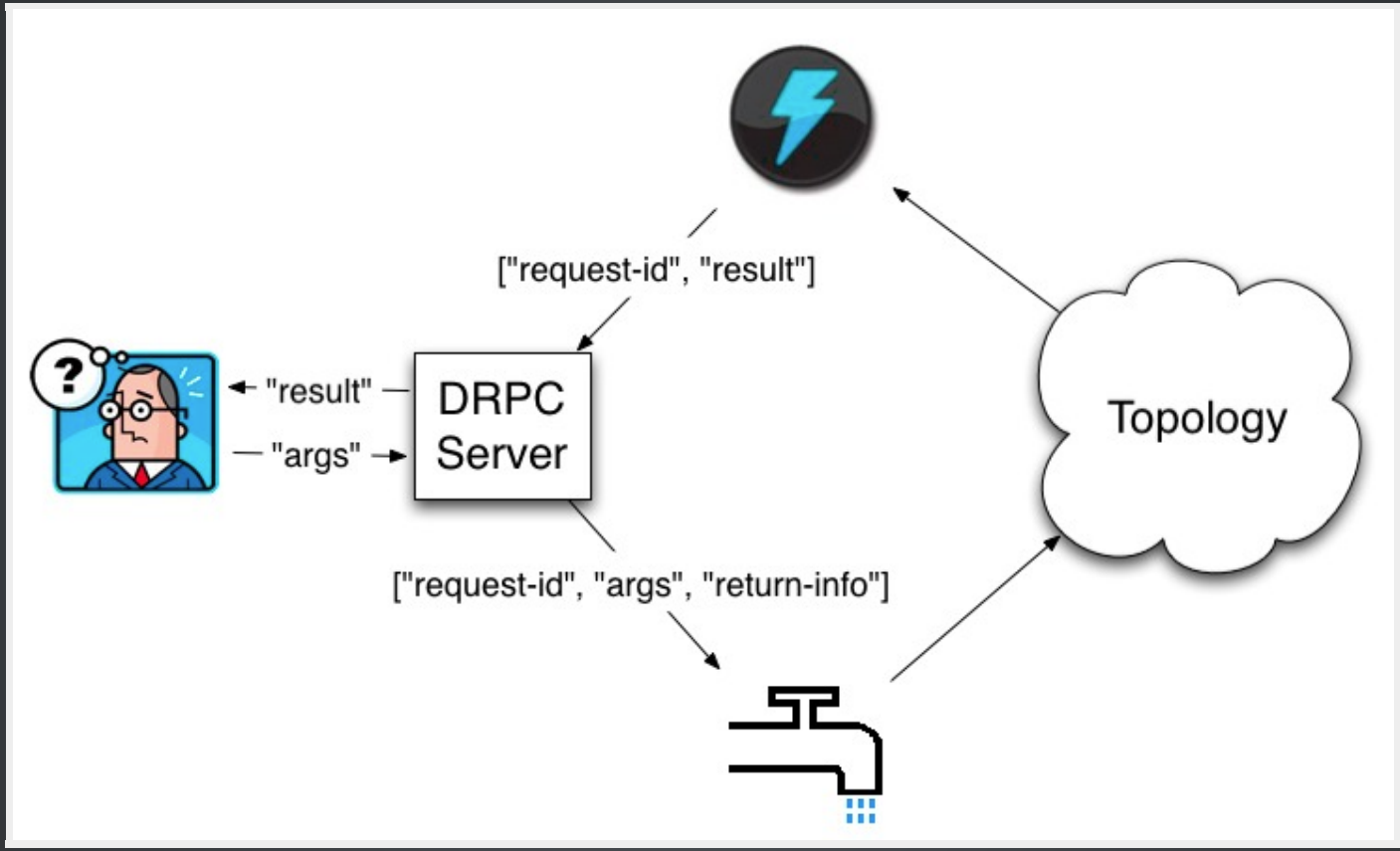
```
public class WordNormalizer implements IRichBolt {
  private OutputCollector collector;
  public void cleanup() {}
  /**
  * The bolt will receive the line from the
  * words file and process it to Normalize this line
  *
  * The normalize will be put the words in lower case
  * and split the line to get all words in this
  */
  public void execute(Tuple input) {
    String sentence = input.getString(0);
    String[] words = sentence.split(" ");
    for(String word : words){
      word = word.trim();
      if(!word.isEmpty()){
        word = word.toLowerCase();
```

WORDCOUNTER

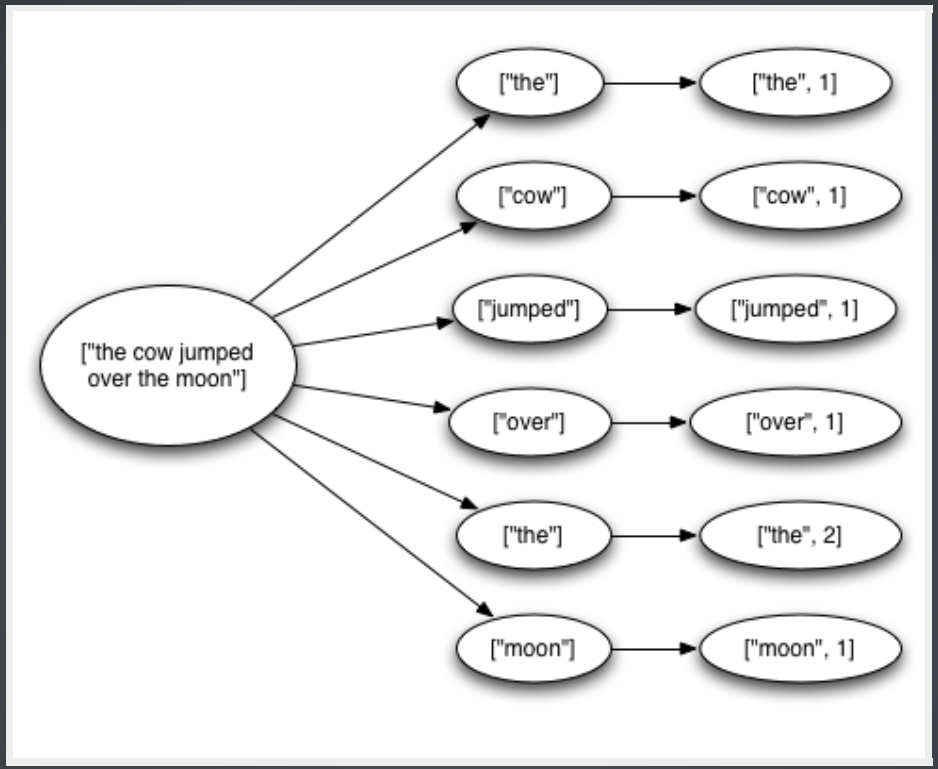
```
public class WordCounter implements IRichBolt {
  Integer id;
  String name;
  Map<string, integer=""> counters;
  private OutputCollector collector;

  /**
  * At the end of the spout (when the cluster is shutdown
  * We will show the word counters
  */
  @Override
  public void cleanup() {
    System.out.println("-- Word Counter ["+name+"-"+id+"] --");
    for(Map.Entry<string, integer=""> entry : counters.entrySet()){
      System.out.println(entry.getKey()+": "+entry.getValue());
    }
  }
```

DRPC


["request-id", "result"]
"result"
"args"
DRPC
Server
Topology
["request-id", "args", "return-info"]

记录级容错


["the cow jumped over the moon"]
["the"]
["the", 1]
["cow"]
["cow", 1]
["jumped"]
["jumped", 1]
["over"]
["over", 1]
["the"]
["the", 2]
["moon"]
["moon", 1]

记录级容错

- Tuple超过topology.message.timeout.secs未被处理被视为失效
- 首先Storm从Spout中请求nextTuple获取tuple
- Spout通过SpoutOutputCollector提供的open方法输出tuple到输出流,并提供一个message id标识此tuple
- 接着tuple被Bolt消费,Storm跟踪它创建的消息树。如果Storm检测到tuple被fully process，Storm调用ack传入该message id。如果tuple超时将在Spout中调用fail方法。

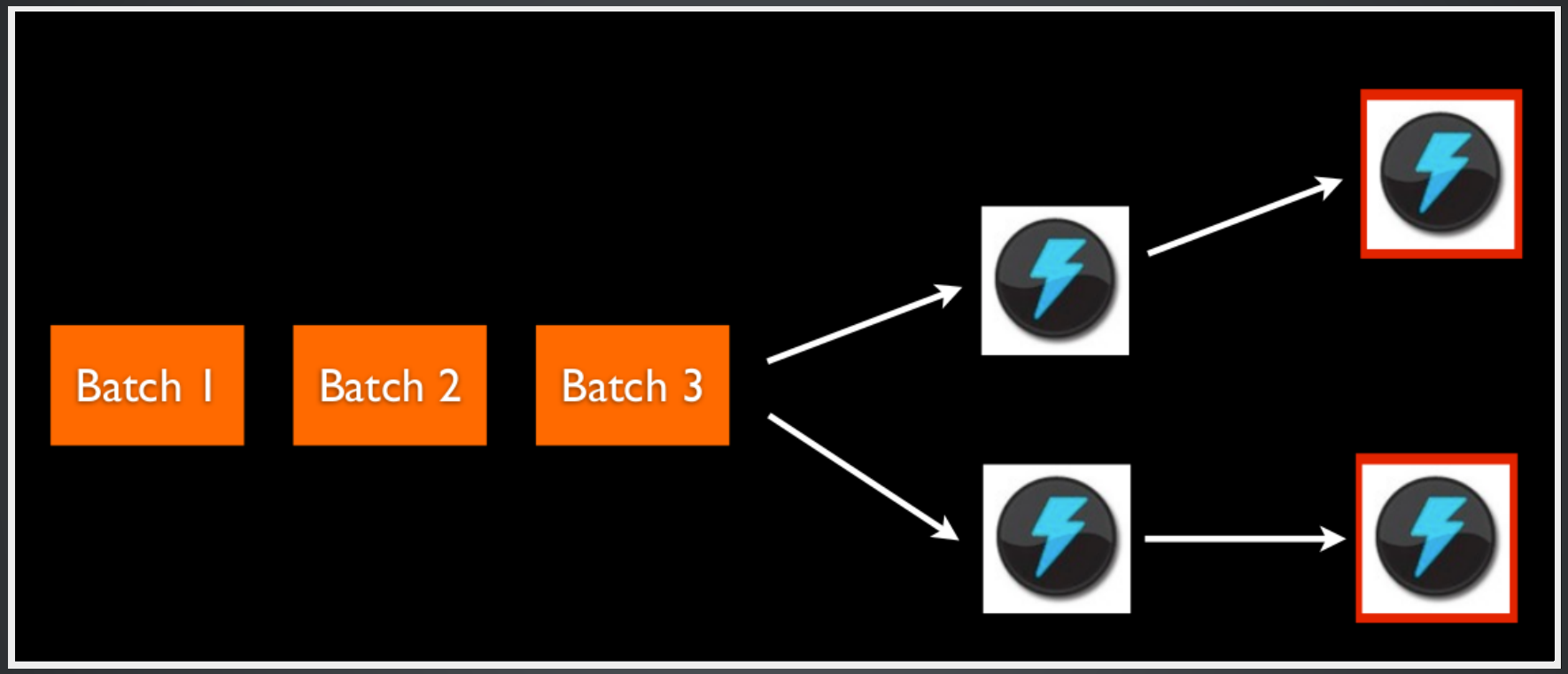
事务拓扑

- Transactional topology
- Trident

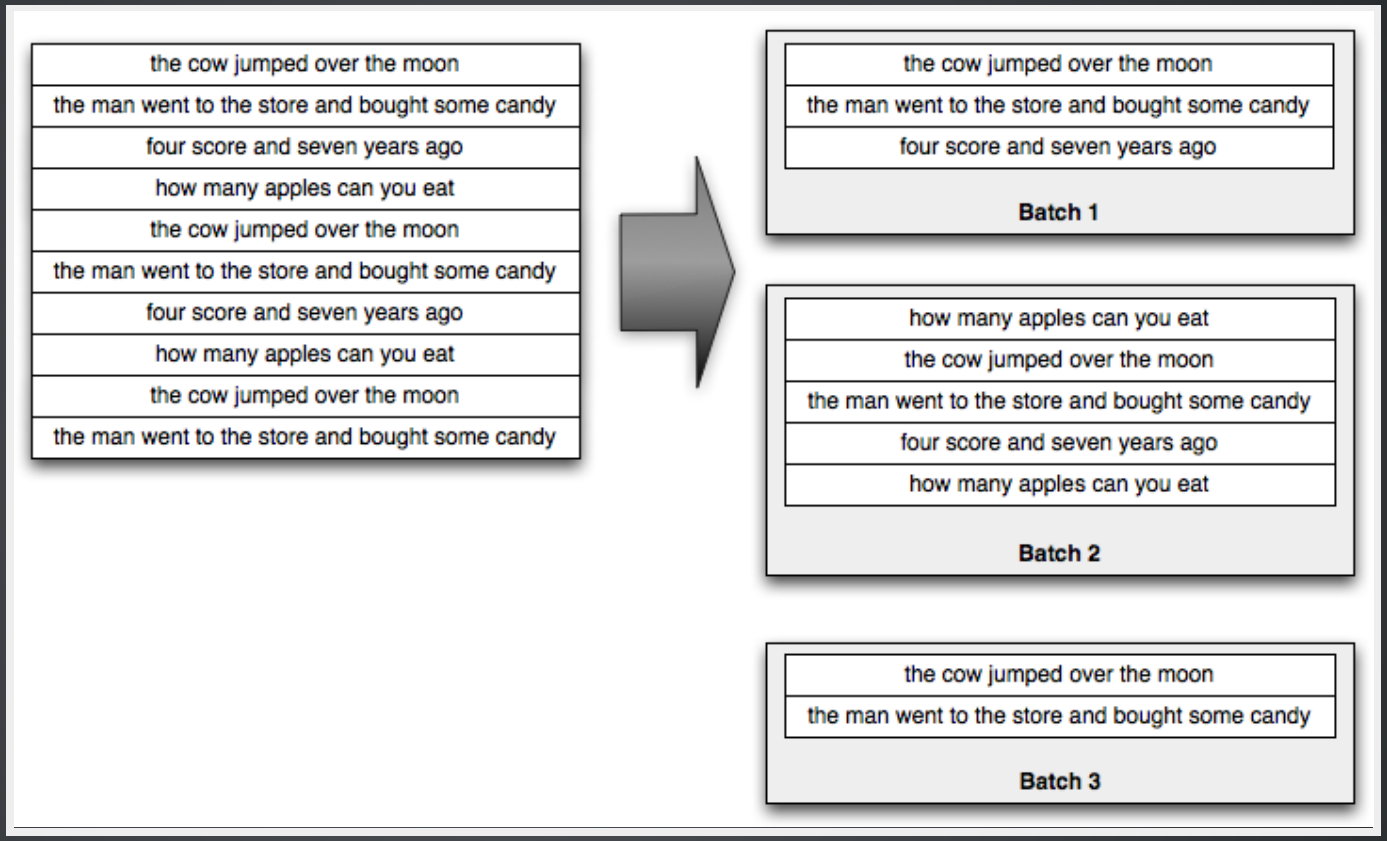
TRANSACTIONAL TOPOLOGY

- 0.7.0引入
- 需要支持指定批次replay的queue
- 实现了基于kafka的transactional spout

TRANSACTIONAL TOPOLOGY



TRIDENT



- 0.8.0引入

TRIDENT SPOUT

```
FixedBatchSpout spout = new FixedBatchSpout(new Fields("sentence"), 3,
                new Values("the cow jumped over the moon"),
                new Values("the man went to the store and bought some canc
                new Values("four score and seven years ago"),
                new Values("how many apples can you eat"),
spout.setCycle(true);
```

TRIDENT TOPOLOGY

```
TridentTopology topology = new TridentTopology();
TridentState wordCounts =
     topology.newStream("spout1", spout)
       .each(new Fields("sentence"), new Split(), new Fields("word"))
       .groupBy(new Fields("word"))
       .persistentAggregate(new MemoryMapState.Factory(), new Count(), ne
       .parallelismHint(6);
```

TRIDENT BASEFUNCTION

```
public class Split extends BaseFunction {
   public void execute(TridentTuple tuple, TridentCollector collector) {
       String sentence = tuple.getString(0);
       for(String word: sentence.split(" ")) {
           collector.emit(new Values(word));
       }
   }
}
```

TRIDENT DRPC

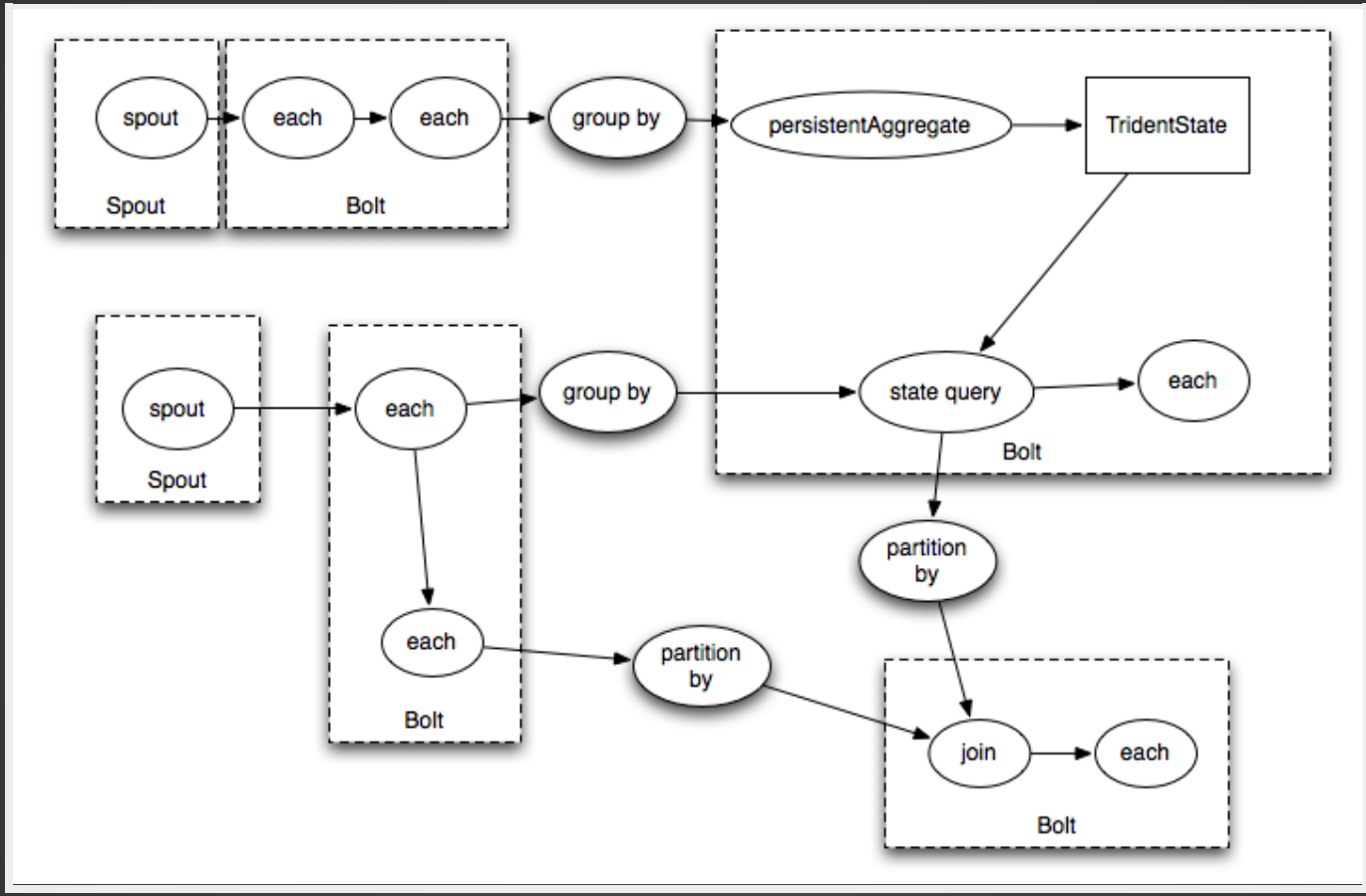
```
DRPCClient client = new DRPCClient("drpc.server.location", 3772);
System.out.println(client.execute("words", "cat dog the man");
// prints the JSON-encoded result, e.g.: "[[5078]]"
```

```
topology.newDRPCStream("words")
       .each(new Fields("args"), new Split(), new Fields("word"))
       .groupBy(new Fields("word"))
       .stateQuery(wordCounts, new Fields("word"), new MapGet(), new Fiel
       .each(new Fields("count"), new FilterNull())
       .aggregate(new Fields("count"), new Sum(), new Fields("sum"));
```

状态存储

- Trident解决的两个问题
  - 为每一批次提供了一个唯一的transaction id，重试时使用一致的transaction id
  - 状态更新依据批次的顺序
- 存储方式
  - Memory
  - Memcached
  - Cassandra
  - ...

TRIDENT TOPOLOGY



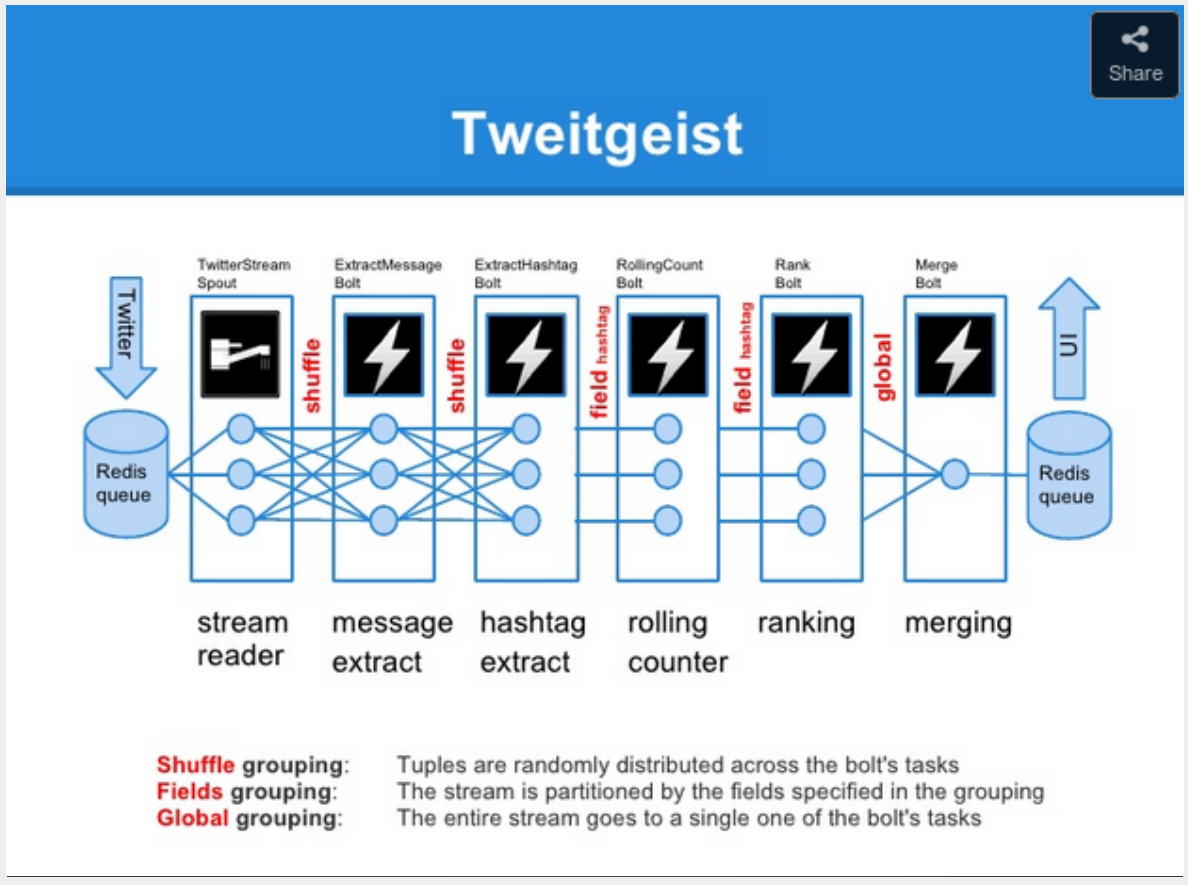
STORM的问题

- 向topology增加业务时需要删除原有topology部署新的topology(0.9增加了swap)
- topology间不能传递数据（只能通过Queue队列间接传递）

业界应用

TWITTER场景(TWEITGEIST)

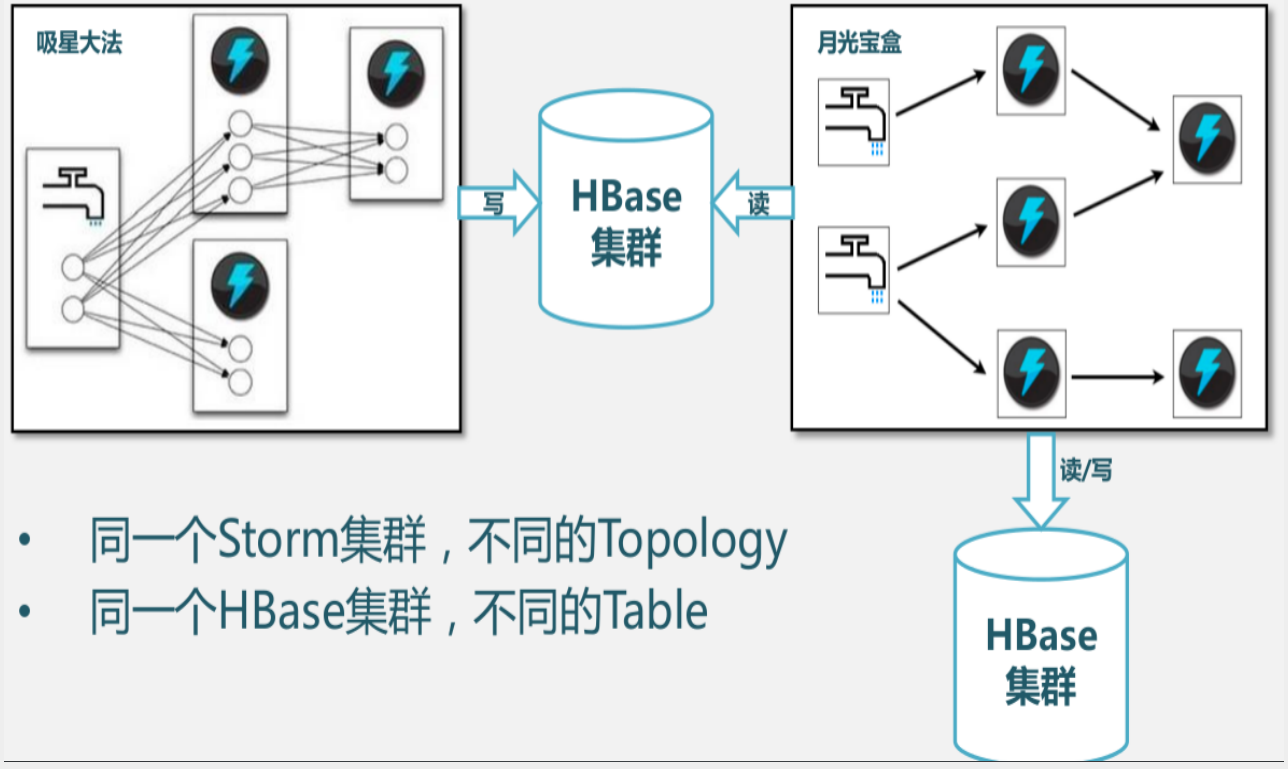
LIVE TOP 10 TRENDING HASHTAGS ON TWITTER



一淘数据部(月光宝盒)

月光宝盒项目简介

一淘 etao.com 数据部
数据让生活更简单!



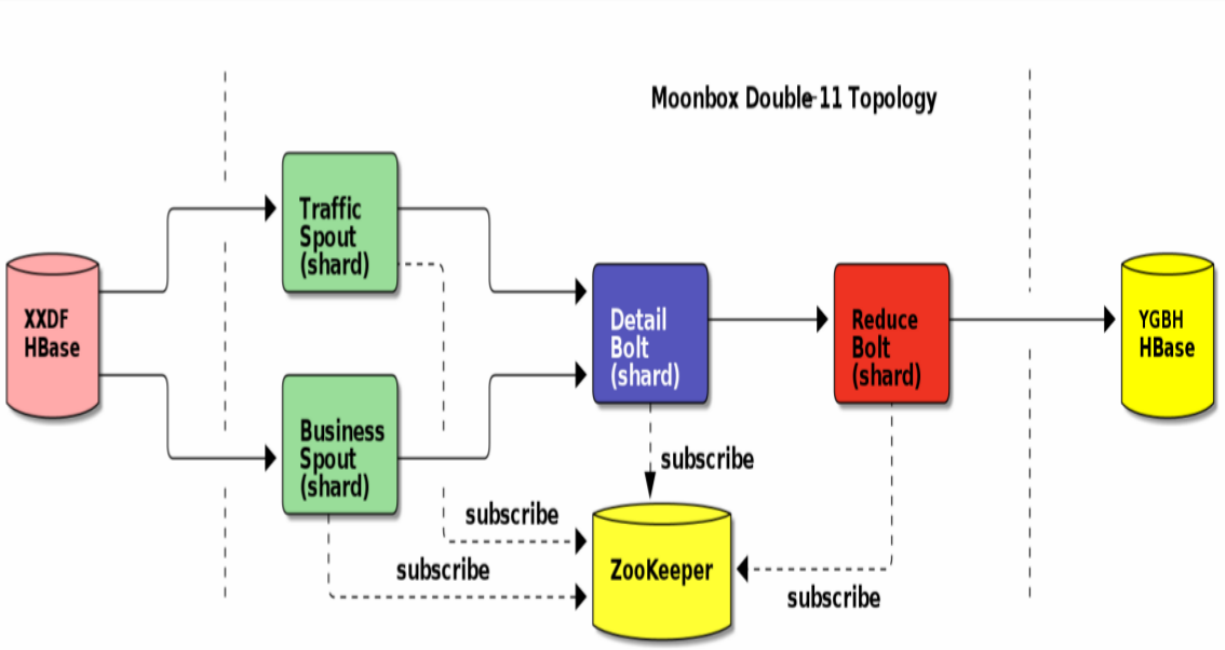
- 同一个Storm集群，不同的Topology
- 同一个HBase集群，不同的Table

一淘数据部(月光宝盒)

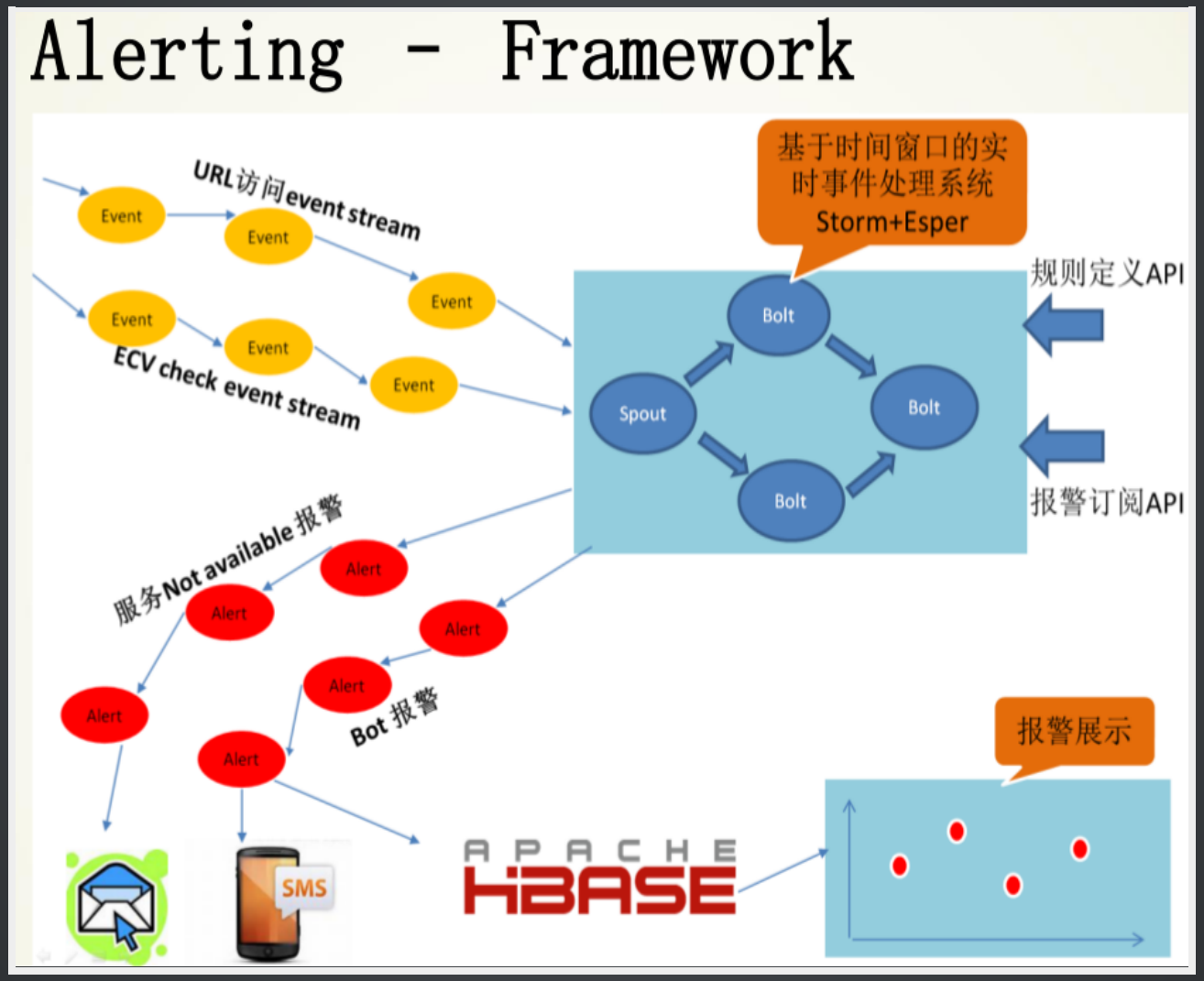
月光宝盒项目简介

一淘 etao.com 数据部
数据让生活更简单!

- 双11拓扑任务



携程(ALERTING)


Alerting - Framework
URL访问event stream
Event
Event
Event
Event
Event
Event
ECV check event stream
基于时间窗口的实时事件处理系统
Storm+Esper
规则定义API
Bolt
Spout
Bolt
Bolt
报警订阅API
服务Not available 报警
Alert
Alert
Alert
Alert
Alert
Alert
Bot 报警
报警展示
SMS
APACHE
HBASE

运维监控

监控工具

- Storm UI
- JMX/VisualVM
- Yammer Metrics
- Ganglia/Graphite
- Log4j
- Nagios(监控硬件、系统、log中的ERROR/WARN)
- OpenTSDB
- storm-monitor

STORM UI

JMX/VISUALVM

YAMMER METRICS

HTTP://METRICS.CODAHALE.COM/

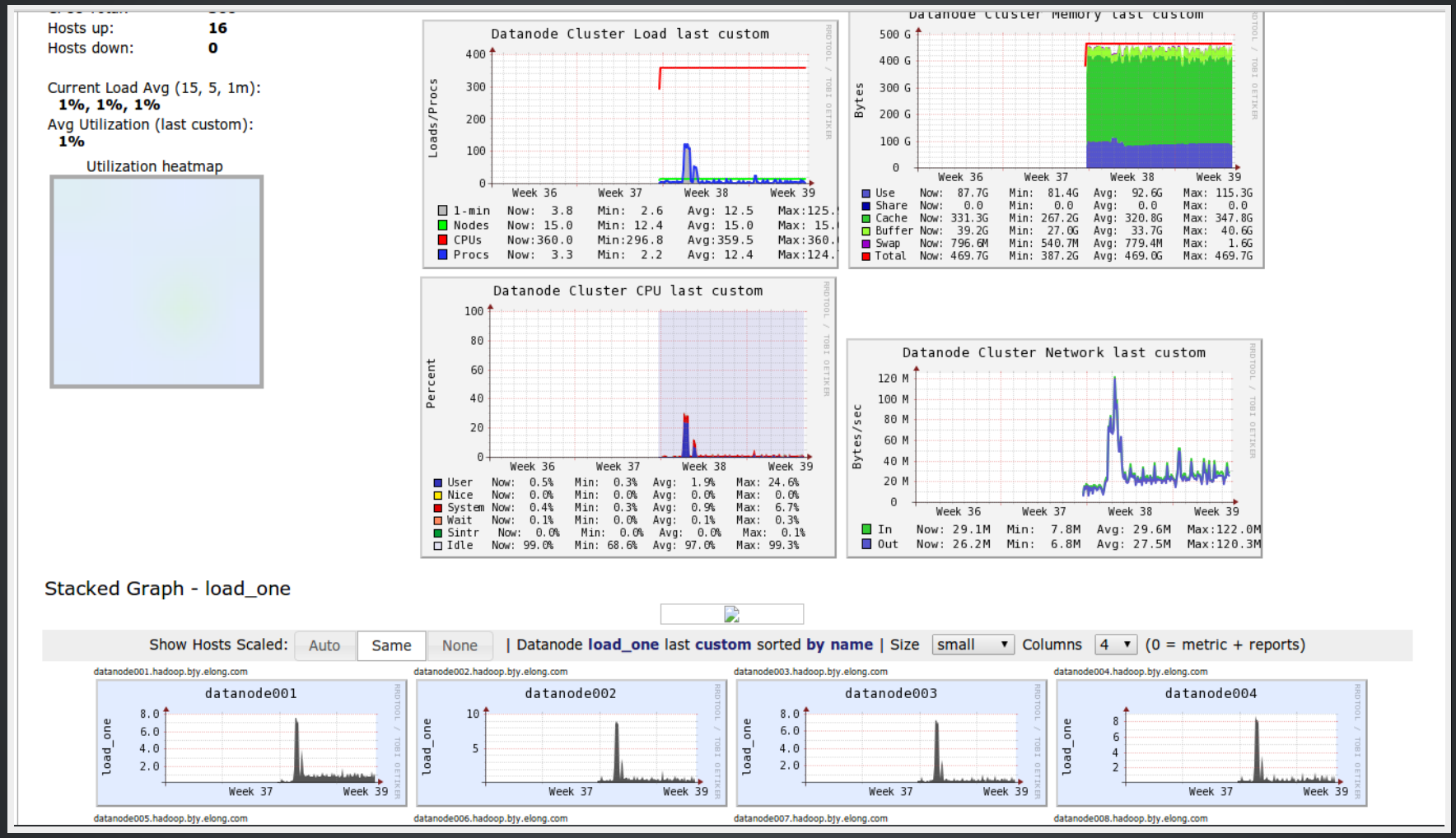
```
<dependencies>
    <dependency>
        <groupId>com.codahale.metrics</groupId>
        <artifactId>metrics-core</artifactId>
        <version>${metrics.version}</version>
    </dependency>
</dependencies>
```

YAMMER METRICS

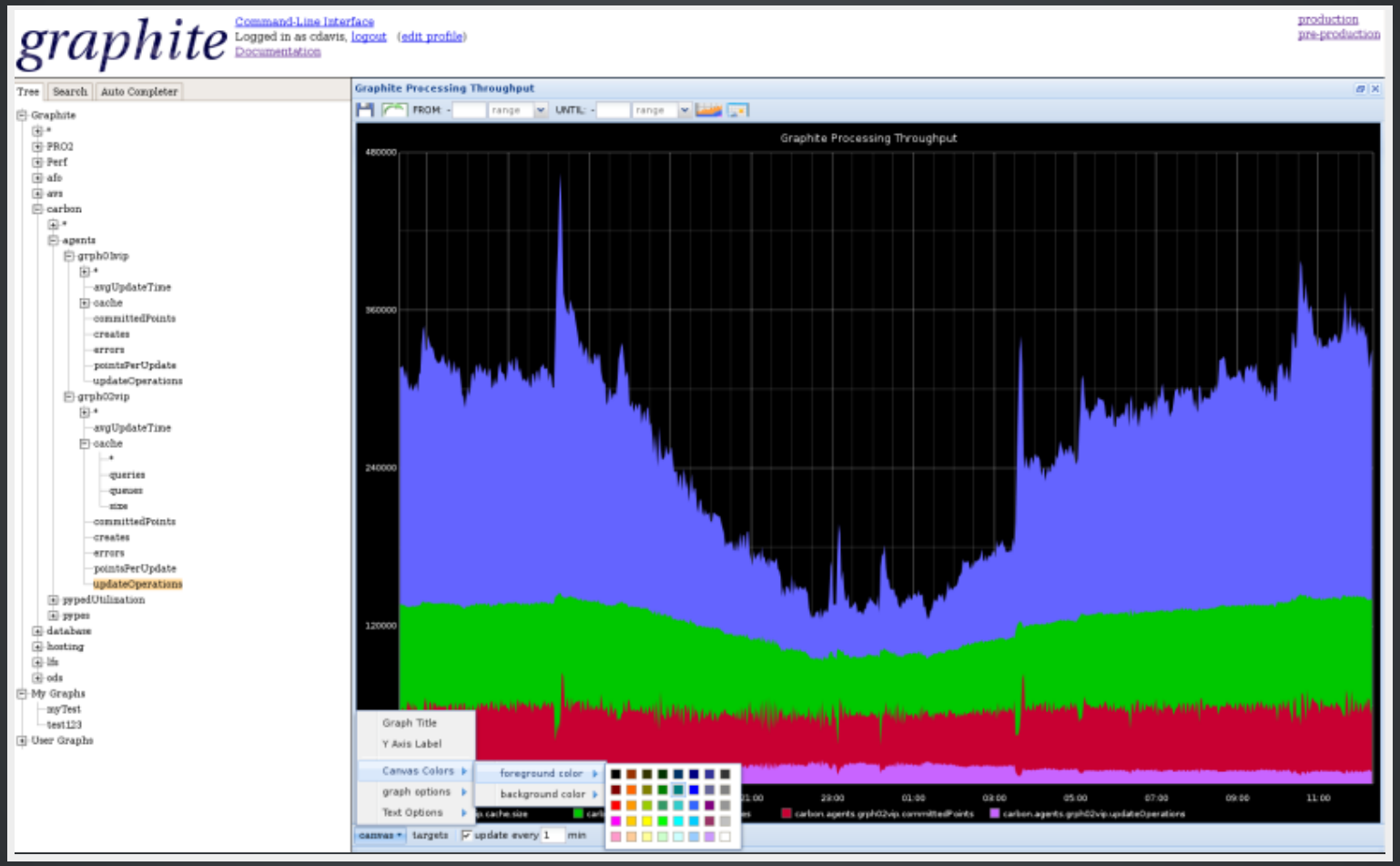
```
//application level static
final MetricRegistry metrics = new MetricRegistry();
//Gauges 标尺
public class QueueManager {
    private final Queue queue;

    public QueueManager(MetricRegistry metrics, String name) {
        this.queue = new Queue();
        metrics.register(MetricRegistry.name(QueueManager.class, name, "s
                         new Gauge<integer>() {
                             @Override
                             public Integer getValue() {
                                 return queue.size();
                             }
                         });
    }
}
```

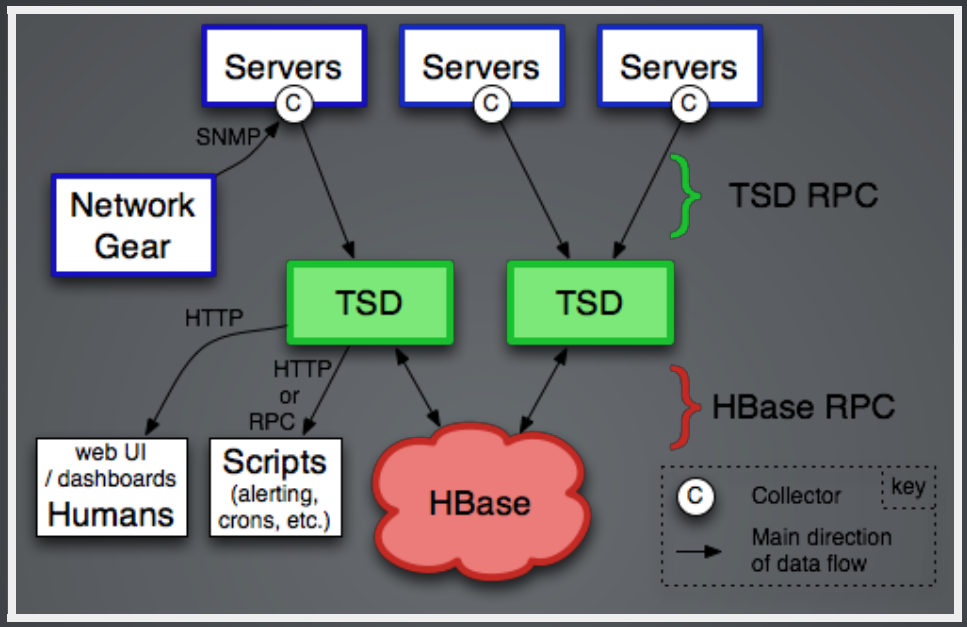
GANGLIA

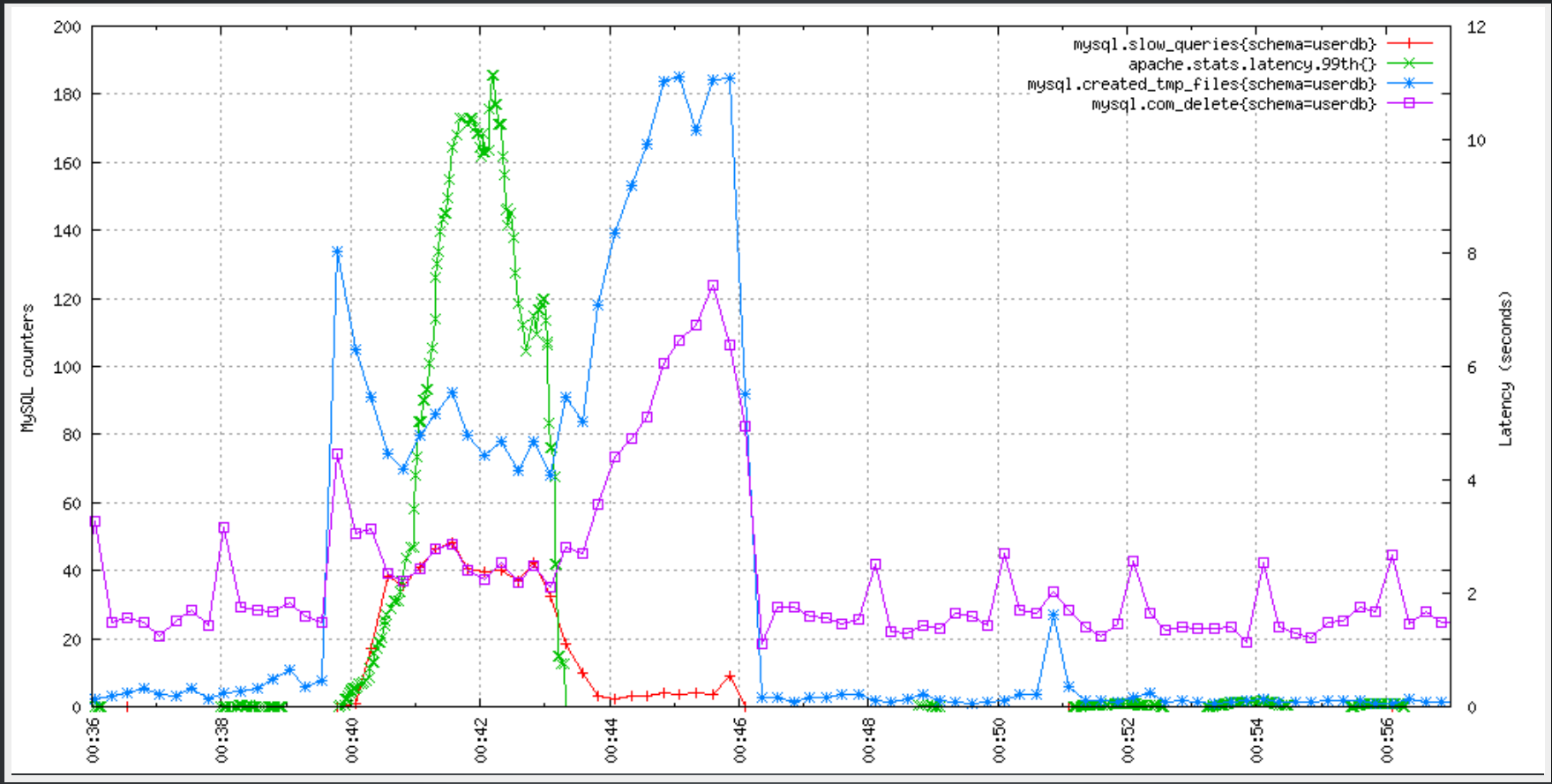

Hosts up: 16
Hosts down: 0
Current Load Avg (15, 5, 1m):
1%, 1%, 1%
Avg Utilization (last custom):
1%
Utilization heatmap
Datanode Cluster Load last custom
Loads/Procs
1-min Now: 3.8 Min: 2.6 Avg: 12.5 Max:125.
Nodes Now: 15.0 Min: 12.4 Avg: 15.0 Max: 15.
CPUs Now:360.0 Min:296.8 Avg:359.5 Max:360.
Procs Now: 3.3 Min: 2.2 Avg: 12.4 Max:124.
Datanode Cluster Memory last custom
Bytes
Use Now: 87.7G Min: 81.4G Avg: 92.6G Max: 115.3G
Share Now: 0.0 Min: 0.0 Avg: 0.0 Max: 0.0
Cache Now: 331.3G Min: 267.2G Avg: 320.8G Max: 347.8G
Buffer Now: 39.2G Min: 27.0G Avg: 33.7G Max: 40.6G
Swap Now: 796.6M Min: 540.7M Avg: 779.4M Max: 1.6G
Total Now: 469.7G Min: 387.2G Avg: 469.0G Max: 469.7G
Datanode Cluster CPU last custom
Percent
User Now: 0.5% Min: 0.3% Avg: 1.9% Max: 24.6%
Nice Now: 0.0% Min: 0.0% Avg: 0.0% Max: 0.0%
System Now: 0.4% Min: 0.3% Avg: 0.9% Max: 6.7%
Wait Now: 0.1% Min: 0.0% Avg: 0.1% Max: 0.3%
Sintr Now: 0.0% Min: 0.0% Avg: 0.0% Max: 0.1%
Idle Now: 99.0% Min: 68.6% Avg: 97.0% Max: 99.3%
Datanode Cluster Network last custom
Bytes/sec
In Now: 29.1M Min: 7.8M Avg: 29.6M Max:122.0M
Out Now: 26.2M Min: 6.8M Avg: 27.5M Max:120.3M
Stacked Graph - load_one
Show Hosts Scaled: Auto Same None | Datanode load_one last custom sorted by name | Size small Columns 4 (0 = metric + reports)
datanode001.hadoop.bjy.elong.com
datanode001
datanode002.hadoop.bjy.elong.com
datanode002
datanode003.hadoop.bjy.elong.com
datanode003
datanode004.hadoop.bjy.elong.com
datanode004
load_one
Week 37
Week 39
datanode005.hadoop.bjy.elong.com
datanode006.hadoop.bjy.elong.com
datanode007.hadoop.bjy.elong.com
datanode008.hadoop.bjy.elong.com

GRAPHITE


graphite
Command-Line Interface
Logged in as cdavis, logout (edit profile)
Documentation
production
pre-production
Tree
Search
Auto Completer
Graphite
*
PRO2
Perf
afo
avs
carbon
*
agents
grph01vip
*
avgUpdateTime
cache
committedPoints
creates
errors
pointsPerUpdate
updateOperations
grph02vip
*
avgUpdateTime
cache
*
queries
queues
size
committedPoints
creates
errors
pointsPerUpdate
updateOperations
pypedUtilization
pypes
database
hosting
lfs
ods
My Graphs
myTest
test123
User Graphs
Graphite Processing Throughput
FROM
UNTIL
Graphite Processing Throughput
480000
360000
240000
120000
Graph Title
Y Axis Label
Canvas Colors
graph options
Text Options
foreground color
background color
carbon.agents.grph02vip.committedPoints
carbon.agents.grph02vip.updateOperations
canvas
targets
update every 1
min

OPENTSDB


Servers
Servers
Servers
C
C
C
SNMP
Network Gear
TSD RPC
TSD
TSD
HTTP
HTTP or RPC
HBase RPC
web UI / dashboards
Humans
Scripts (alerting, crons, etc.)
HBase
key
C
Collector
Main direction of data flow


mysql.slow_queries{schema=userdb}
apache.stats.latency.99th{}
mysql.created_tmp_files{schema=userdb}
mysql.com_delete{schema=userdb}
MySQL counters
Latency (seconds)
0
20
40
60
80
100
120
140
160
180
200
0
2
4
6
8
10
12
00:36
00:38
00:40
00:42
00:44
00:46
00:48
00:50
00:52
00:54
00:56

STORM-MONITOR

HTTPS://GITHUB.COM/KILLME2008/STORM-MONITOR参考用于监控NIMBUS

- 监控supervisor数目是否正确，当supervisor挂掉的时候会发送警告
- 监控nimbus是否正常运行，monitor会尝试连接nimbus，如果连接失败就认为nimbus挂掉。
- 监控topology是否正常运行，包括它是否正常部署，是否有运行中的任务

其它可选监控工具

- Ooyala metrics_storm
- Storm 0.9 Metrics
  - 按固定的时间窗口收集任意的用户自定义metric
  - metric最终被Storm聚合
  - 可自定义MetricsConsumer

服务性能监控

- 拓扑图
  - 吞吐量
  - 延迟
- Bolt和spout
  - 吞吐量
  - 延迟
  - 队列时间
  - 执行时间
- 数据来源和数据存储
  - 读/写条数
  - 读/写速度
- Storm 0.8.2 new metrics
  - process latency(tuple直到被ack的时间)
  - execute latency(tuple被调度执行的时间)
  - capacity(上十分钟bolt用于执行tuple的时间百分比)

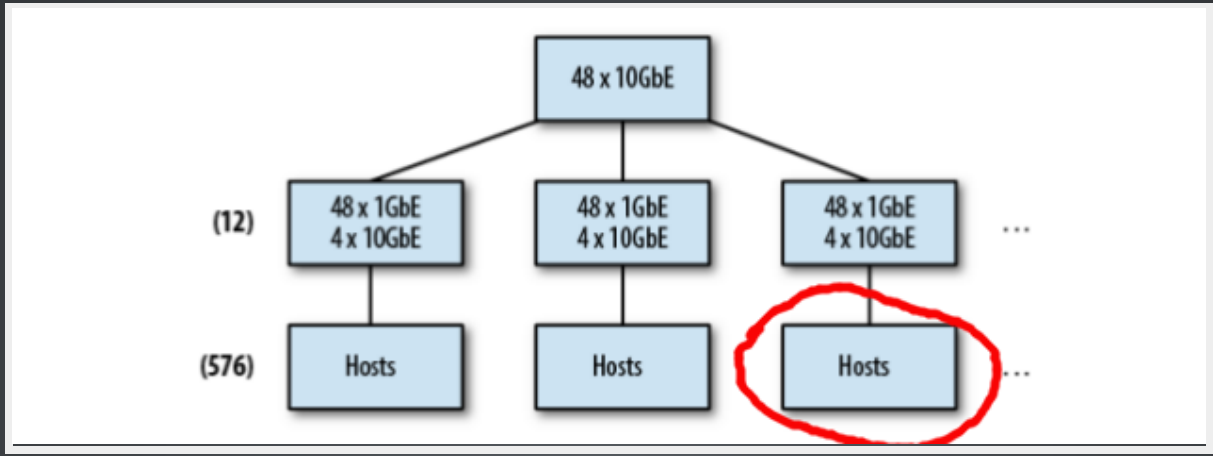
CHECKLIST应用级监控

- Kafka读取速度
- HBase写入速度
- 规则处理引擎某topic单条数据处理时间
- 规则处理引擎处理速度
- 规则处理引擎出错次数
- 处理数据延迟(根据数据业务日期)
- QueryTools查询响应时间
- 数据接口监控

推荐监控方案

- 服务或组件监控
  - Nagios(硬件、服务进程、log)参考storm_monitor
  - Storm UI(Thrift)
  - Ooyala metrics_storm
  - Ganglia/Graphite
- 应用监控
  - Yammer Metrics

网络规划


48 x 10GbE
(12)
48 x 1GbE
4 x 10GbE
48 x 1GbE
4 x 10GbE
48 x 1GbE
4 x 10GbE
...
(576)
Hosts
Hosts
Hosts
...

THE END